```
// objects.cpp

#include "stdafx.h"

//--------------------------------------------------------------------

const char *uniqueNames[] = {
   "Unknown", "No", "Unlikely", "Likely", "Yes"
};

const char *browserNames[] = {
   "Unknown",
   "Netscape",
   "NCSA Mosaic",
   "AOL Browser",
   "HotJava",
   "Microsoft",
   "OmniWeb",
   "Lynx",
   "NetCruiser",
   "IBM WebExplorer",
   "AIR Mosaic/Spry Mosaic",
   "MacWeb",
   "NetManage Chameleon",
   "NetSurfer",
   "Enhanced Mosaic",
   "eworld Browser",
   "Prodigy Browser",
   "Delphi Browser",
   "CNN Browser",
   "InterNotes",
   "Wollagong/ATM Emissary",
   "PipeMacWeb",
   "InternetMCI",
   "Quarterdeck Mosaic"
};

const char *osNames[] = {
   "Unknown",
   "Win16",
   "Win32",
   "Windows",
   "Win95",
   "WinNT",
   "OS/2",
   "Macintosh",
   "Mac 68k",
   "Mac PowerPC",
   "Unix (brand unknown)",
   "Unix (other)",
   "Unix (Sun)",
   "Unix (Linux)",
   "Unix (HP)",
   "Unix (AIX)",
   "Unix (OSF)",
   "Unix (IRIX)",
   "NEXT",
   "Unix (SGI)"
};

const char *domainTypeNames[] = {
   "Unknown",
   "Commercial", "Education", "Government",
   "Military", "K-12", "Foreign", "Networks",
   "Organizations",

   0,

   "AOL",
   "Prodigy",
   "Compuserve",
   "Delphi",
   "eworld",
   "MSN",
   "DowJones",
```



```
   "Genie",

   0,0,0,0,0,0,0,0,
   "Reserved for ISP Names"
};

const char *ISPNames[] = {
   "ISP",
   "NetCom",
   "PSI",
   "UUNet",
   "Advantis",
   "Concentric Research Corp.",
   "CRL",
   "MCI",
   "Portal Information Network"
};

const char *salesStr[] = {
   "unknown",
   "$1 - $49,999",
   "$50,000 - $99,999",
   "$100,000 - $249,999",
   "$250,000 - $499,999",
   "$500,000 - $999,999",
   "$1 million - $4,999,999",
   "$5 million - $9,999,999",
   "$10 million - $49,999,999",
   "$50 million - $499,999,999",
   "$500 million - $999,999,999",
   "$1 billion and over"
};

const char *empStr[] = {
   "unknown",
   "1 - 4",
   "5 - 9",
   "10 - 14",
   "15 - 19",
   "20 - 49",
   "50 - 99",
   "100 - 499",
   "500 - 999",
   "1,000 and over",
};

const char *genderStr[] = {
   "unknown",
   "Male",
   "Female"
};

const char *timesStr[] = {
   "12am-1am",
   "1am-2am",
   "2am-3am",
   "3am-4am",
   "4am-5am",
   "5am-6am",
   "6am-7am",
   "7am-8am",
   "8am-9am",
   "9am-10am",
   "10am-11am",
   "11am-12pm",
   "12pm-1pm",
   "1pm-2pm",
   "2pm-3pm",
   "3pm-4pm",
   "4pm-5pm",
   "5pm-6pm",
   "6pm-7pm",
   "7pm-8pm",
   "8pm-9pm",
   "9pm-10pm",
```

```
   "10pm-11pm",
   "11pm-12am"
};

const char *dowStr[] = {
   "Sunday",
   "Monday",
   "Tuesday",
   "Wednesday",
   "Thursday",
   "Friday",
   "Saturday"
};

#if !defined(_JUSTSTRINGS)

#include <strstrea.h>
#include <fstream.h>
#include "winsock.h"
#include "objects.h"
#include "tables.h"
#include "/d/toolkit/isf_util.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/dbutil.h"
#include "/d/derive/sqlderive.h"
#include "/d/newderive/sig.h"
#include "rememberad.h"

extern ofstream errLog;

extern Ad *badKeyErrorAd;

int nextAd = 0;

#if !defined(_DERIVE)
int nAds();
#endif
extern Ad *defaultAd;

//---------------------------------------------------------------------
// User

BOOL User::cookieCapable() const
{
   // todo: add new version of Internet Explorer

   return browser == brNetscape &&
      (bVer2 >= 1 ||
      bVer1 > 1);
}

void User::deriveDomainTypeFromBrowser()
{
   switch( browser ) {
      case brAOL:
         domainType = dtAOL;
         break;
      case brEWorld:
         domainType = dtEWorld;
         break;
      case brProdigy:
         domainType = dtProdigy;
         break;
      case brDelphi:
         domainType = dtDelphi;
         break;
      default:
         ;
   };
}

DWORD User::getID() const
{
```



```
   return userID;
}

User::User()
{
   timedOut = FALSE;
   userID = 0;
   uniqueness = uUnknown;
   ip = 0;
   browser = brUnknown;
   bVer1 = bVer2 = 0;
   os = osUnknown;
   domainType = dtUnknown;
// for( int i = 0; i < MAXSICS; i++ )
//    sicCodes[i] = 0;
   nEmployees = 0;
   salesVolume = 0;
   proxy = FALSE;
   isNetworkDescription = FALSE;
   ftpTried = FALSE;
   hasCookie = FALSE;
}

void User::describe(Database& db, strstream& text)
{
   in_addr ipAddr = (in_addr&) ip;
   text << "<b>ip:          </b>"
        << (int) ipAddr.S_un.S_un_b.s_b1 << '.'
        << (int) ipAddr.S_un.S_un_b.s_b2 << '.'
        << (int) ipAddr.S_un.S_un_b.s_b3 << '.'
        << (int) ipAddr.S_un.S_un_b.s_b4 << "\r\n";

   gender = ' ';
   if( !emailAddr.isEmpty() ) {
      // get name/gender
      Cursor c(db);
      CString g,f,l;
      c.bind(g);
      c.bind(f);
      c.bind(l);
      Signature e;
      e.email = emailAddr;
      e.l.pullEmail();
      if( !e.l.domain.isEmpty() ) {
         char sql[1024] = "select gender,fname1,lname1 from listings where emailname=";
         addValue(sql, e.l.emailname, FALSE);
         strcat(sql, " and domain=");
         addValue(sql, e.l.domain, FALSE);

         c.exec(sql);
         if( c.fetchNext() ) {
            if( g.GetLength() )
               gender = g.GetAt(0);
            name = f + ' ' + l;
            cap(name);
         }
      }
      db.commit();
   }

   text << "<b>unique:      </b>" << uniqueNames[uniqueness] << "\r\n";
   text << "<b>cookie:      </b>" << (hasCookie ? "Yes" : "No") << "\r\n";
   text << "<b>browser:     </b>" << browserNames[browser] << "\r\n";
   text << "<b>browser ver: </b>" << (int) bVer1 << '.' << (int) bVer2 << "\r\n";
   text << "<b>os:          </b>" << osNames[os] << "\r\n";
   text << "<b>domain type: </b>" << domainTypeNames[domainType] << "\r\n";
   text << "<b>proxy:       </b>" << (proxy ? "Yes" : "No") << "\r\n";
   text << "<hr>";
   text << "<b>name:        </b>" << name << "\r\n";
   text << "<b>title:       </b>" << title << "\r\n";
   text << "<b>state:       </b>" << location.state << "\r\n";
   text << "<b>zip code:    </b>" << location.zipCode << "\r\n";
   text << "<b>area code:   </b>" << location.areaCode << "\r\n";
   text << "<b>phone:       </b>" << phone << "\r\n";
   text << "<b>e-mail:      </b>" << emailAddr << "\r\n";
```

```
   Ad *ad = Ad::findSentTo(user, from);

   if( ad == 0 ) {      // fix!
      delete user;
      return;
   }

   SitePage *page = SitePage::lookupPage(from, request);

   CString hdr =
      "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\nContent-Length: ";
   char buf[32000];
   *buf = 0;
   strstream text(buf, 32000, ios::out);

   // fill content
   text << "<html><body><h1>Jump Redirect</h1>";
   text << "<pre>";
   text << "jumping from document: " << from << "\r\n";
   text << "        would jump to: "
        << "<a href=\"" << (const char *) ad->jumpTo << "\">"
        << (const char *) ad->jumpTo << "</a>\r\n\r\n";
   text << "                  ad: " << (const char *) ad->fullName() << "\r\n\r\n";

   CString fn = ad->fileName;
   text << "<center><img src=\"/"
        << (const char *) fn
        << "\">";
   text << "</pre></body></html>";

   int n = text.pcount();
   char temp[100];
   itoa(n, temp, 10);   // content length
   hdr += temp;
   hdr += "\r\n\r\n";

   c->write( (const char *) hdr, hdr.GetLength() );
   c->write(buf, n);

   logJump(ad, user, page);

   delete page;
   delete ad;
   delete user;
*/
}

void GetRequest::sendFrame(const char *from)
{
   CString s = "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n";
   s += "\r\n<HTML><BODY><CENTER><a href=\"http://206.4.219.5/jump/";
   s += from;
   s += "\">\n<img src=\"http://206.4.219.5/ad/";
   s += from;
   s += "\">\n</a></center></html>";   //  Width=468 Height=60
   c->write( (const char *) s, s.GetLength() );
}

void GetRequest::activity(const char *activityStr)
{
   // go ahead and send for best response time
   sendFile("c:\\ian\\html\\dot.gif");

   BOOL bad = FALSE;

   // send the file first
   ActivityType type;
   CString siteKey;
   BOOL ok = TRUE;
   switch( *activityStr )
   {
      case 'a':
         type = Interest;
         break;
      case 'i':
```



```
         type = InfoRequest;
         break;
      case 's':
         type = Sale;
         break;
      default:
         ok = FALSE;
   }

   if( ok ) {
      const char *p = activityStr + 1;
      if( *p != '/' )
         ok = FALSE;
      else {
         p++;
         const char *q = strchr(p, '/');
         if( q == 0 )
            ok = FALSE;
         else
            siteKey = CString(p, q - p);
      }
   }

   if( ok ) {
      Database *db = getFromPool();
      User *user = User::lookupUser(*db, userIP, request);
      DWORD advertiserID = 0;
      // todo: fix if not assigned a user ID, (use IP)?
      if( user->userID != 0 ) // if not from LAN, skip logging
      {
         Cursor c(*db);
         c.bind(SQL_C_LONG, &advertiserID, sizeof(advertiserID));
         char sql[1024] = "select id from advertisers where sitekey=";
         addValue(sql, siteKey, FALSE);
         c.exec(sql);
         ok = c.fetchNext();
      }

      db->commit();

      if( ok ) {
         ++activity;
         if( advertiserID != 0 )
            logActivity(user, advertiserID, type);
      }

      delete user;
      releaseToPool(db);
   }

   if( !ok ) {
      message( CString("Invalidate activity str: ") +
               CString(activityStr).Left(80) );
//    sendError(c, "404 Not Found");
   }
}

void GetRequest::sendAd(const char *from)
{
   if( from && strnicmp(from, "www.", 4) == 0 )
      from += 4;

   Database *db = getFromPoolTimeout();

   static DWORD lastFTP;
   startLatency = GetTickCount();

   User *user;
   SitePage *page;
   Ad *ad;

   user = User::lookupUser(*db, userIP, request, TRUE, TRUE);
   if( db == 0 ) {
      page = 0;
   }
```

```
   else {
      page = SitePage::lookupPage(*db, from, request);
   }
   ad = Ad::getAd(*db, user, page, v == GET);

// if( v == GET) {
//    TRACE("get %s\n", (from);
// }

   static int randCutoff = 0; //RAND_MAX / 4;

   BOOL doFTP = !user->tempUserObject() &&
      !user->ftpTried && user->uniqueness >= uLikely && !user->proxy &&
      rand() < randCutoff && (startLatency - lastFTP > 6000);
   DWORD dw;
   if( doFTP )
      dw = WaitForSingleObject(ftpMutex, 0);
   if( doFTP && dw != WAIT_FAILED && dw != WAIT_TIMEOUT ) {

      lastFTP = startLatency;

      // Remember that we're doing FTP for user.  Only do once.
      user->ftpTried = TRUE;
      user->updateFTPTried(*db);

      // Redirect
      CString s = "Location: ";
      s += "ftp://206.4.219.6/";
      char buf[10];
      wsprintf(buf, "%lx", user->getID());
      s += buf;
      s += '/';
      CString fn = ad->getFileName();
      s += (const char *) fn;

      errLog << "Trying FTP\n";
      errLog << "  user " << user->getID() << '\n';
      errLog << "  browser " << browserNames[(int) user->browser] << '\n';
      errLog << "  url " << s << '\n';

      s += "\r\n";
      sendError(c, "302 Moved Temporarily", s);

      VERIFY( ReleaseMutex(ftpMutex) );

      logAdSend(ad, user, page);

      errLog.flush();

      db->commit();
      releaseToPool(db);
   }
   else
   {
      //f.cs.leave();
      send(*db, ad, user);    // this function calls releaseToPool()
      //f.cs.enter();
      if( v == GET ) {
         static int counter;
         if( ++counter & 2 )
            ad->calcSI();     // update SI every 4 or so deliveries

         rememberSend(ad, user, from);
         logAdSend(ad, user, page);
         if( user->timedOut ) {
            if( db == 0 )
               poolTimeOuts++;
            else
               timeOuts++;
         }

         // state
         c->close();    // flush send
         DWORD endSend = GetTickCount();
         la      s(endLatency - startLatency);
```



```
         adSendTimeMax(endSend - startLatency);
      }
   }

// delete ad;
   delete page;
   delete user;
}

void GetRequest::takeJump(const char *_from)
{
   Database &db = *getFromPool();

// jumpingWhere(from);
// return;

   User *user = User::lookupUser(db, userIP, request, FALSE);

   if( _from && strnicmp(_from, "www.", 4) == 0 )
      _from += 4;

   CString from;
   {
      const char *p = strchr(_from, '?');
      if( p == 0 ) {
         from = _from;
         char buf[512];
         wsprintf(buf, "no ismap %d: %s", user == 0 ? 999 : (int) user->browser, (const char *
         message(buf);
      }
      else
         from = CString(_from, p - _from);
   }

   Ad *ad = Ad::findSentTo(user, from);
   SitePage *page = SitePage::lookupPage(db, from, request);

   ///f.cs.leave();
   CString s = "Location: ";
   s += ad->jumpTo(); // + "?from=iaf";
   s += "\r\n";
   sendError(c, "301 Moved Permanently", s);
   c->close();
   ///f.cs.enter();

   // Must do this so activity will be logged properly.
   // See GetRequest::activity().
   user->makePermanent(db);

   logJump(ad, user, page);

   delete page;
   delete ad;
   delete user;
   db.commit();
   releaseToPool(&db);
}
```

```
// getrequest.cpp
//
#include "stdafx.h"
#include <strstrea.h>
#include <fstream.h>
#include "/d/toolkit/sock.h"
#include "getrequest.h"
#include "rememberad.h"
#include "/d/toolkit/iaf_util.h"
#include "log.h"
#include "status.h"
#include "/d/toolkit/crit.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/dbutil.h"
#include "/d/toolkit/dbpool.h"

extern CriticalSection fast;
//extern Database isfmain;

extern ofstream errLog;
extern int activity;

extern const char *browserNames[];

const char *progName = "AdSvr";

void message(const char *);

void recalcSI();

DWORD startLatency, endLatency;

// This used to prevent multiple concurrent FTP
// requests right now because our FTPD implementation
// only does one at a time.
//
extern HANDLE ftpMutex;

void GetRequest::service()
{
   const char *p = strchr(request, ' ');
   if( p )
      fileName = CString(request, p - request);
   else
      fileName = request;

   if( fileName.Left(4) == "/ad/" )
      sendAd((const char *) fileName + 4);
   else if( fileName.Left(9) == "/adframe/" )
      sendFrame((const char *) fileName + 9);
   else if( fileName.Left(6) == "/jump/" )
      takeJump((const char *) fileName + 6);
   else if( fileName.Left(10) == "/activity/" )
      activity((const char *) fileName + 10);
   else if( fileName.Left(7) == "/whoami" ) {
      //Crit c(fast);
      whoAmI();
   }
   else if( fileName.Left(8) == "/viewad/" ) {
      CString szFileName;
      szFileName.Format( "c:/ian/ads/%s", (LPCTSTR)fileName+8);
      sendFile( szFileName );
   }
   else if( fileName.Left(11) == "/state.htm?" ) {
      sendError(c, "404 Not Found: Results forecast moved to another server");
      //state((const char *) fileName + 11);
   }
   else if( fileName.Left(10) == "/sendinfo/" ) {
      sendInfo((const char *) fileName + 10);
      return;
   }
   else if( fileName.Left(4) == "/si_" ) {
      // send info stuff
      si_((const char *) fileName + 4);
   }
   else if( fileName.Left(9) == "/sysstate" ) {
      sysState();
   }
   else {
      const char *p = fileName;
      if( strnicmp(p, "/java/", 6) == 0 ) {
         if( strstr(p, "..") == 0 )
            sendFile(p);
         else
            sendError(c, "404 Not Found");
      }
      else {
         if( *p == '/' )
            p++;
         if( *p == 0 ) {
            // send default
            sendFile("c:\\ian\\html\\default.htm");
            return;
         } else {
            if( //strchr(p, '/') == 0 && strchr(p, '\\') == 0 &&
               strstr(p, "..") == 0 )
            {
               CString f = "c:\\ian\\html\\";
               f += p;
               sendFile(f);
               return;
            }
         }
         sendError(c, "404 Not Found");
      }
   }

   // Normally we adjust SI for an ad as it is delivered.
   // However, occasionally should do all ads in case one hasn't
   // been delivered but time has passed.
   static int counter;
   if( ++counter > 200 ) {    // adjust constant as traffic increases
      counter = 0;
      Crit c(fast);
      if( allFree() ) {
         recalcSI();    // recalc SI for all ads
      }
      else {
         counter = 175;    // try again soon
      }
   }
}

const char cHeader[] =
   "HTTP/1.0 200 OK\r\nContent-Type: image/gif\r\nContent-Length: ";

// send() should commit the DB if it does any DB operations because
// the caller commits ahead of time so that the transaction won't
// remain open while the file is sent.
//
void GetRequest::send(Database& db, Ad *ad, User *u)
{
   CString hdr = cHeader;

   const BUFSIZE = 32000;
   char buf[BUFSIZE];

   Cookie sendCookie;
   if( &db != 0 ) {
      if( u->hasCookie ) {
         if( u->cookieCapable() && !u->timedOut )
         {
            // If a user record already exists, it's probably because
            // this IP address is shared with other users (proxy, IP pool,
            // etc.)  So, we want to create another record; we don't want
            // to assign the same cookie to different people!
            u->userID = 0; // create new record

            // generate a cookie for the user
```

```
            u->hasCookie = TRUE;
            u->makePermanent(db);
            sendCookie.value = u->getID();
         }
      }

      // release DB here so that we don't keep a db connection occupied
      // while sending the ad
      db.commit();
      releaseToPool(&db);
   }

   CFile f;
   int n = 0;
   if( v == GET ) {
      CString s = ad->fullName();
      if( !f.Open(s, CFile::modeRead | CFile::shareDenyWrite) ) {
         message( CString("couldn't open ") + s );
         TRACE("couldn't open %s\n", (const char *) s);
         ASSERT(FALSE);
         return;
      }

      n = f.Read(buf, BUFSIZE);
      ASSERT( n != 0 && n != BUFSIZE );
   }
   else {
      n = getFileSize( ad->fullName() );
      // next line is a test for NCSA Mosaic HEAD
      //n = 1;
   }

   char temp[100];
   itoa(n, temp, 10);   // content length
   hdr += temp;
   if( !sendCookie.isNull() ) {
      wsprintf(temp,
            "\r\nSet-Cookie: IAF=%lx; path=/; expires=Wed, 09-Nov-99 23:59:00 GMT",
            sendCookie.value);
      hdr += temp;
   }

   // last-modified time
   hdr += "\r\nLast-Modified: " + curHTTPTime();

   //test
// hdr += "\r\nPragma: no-cache";

   hdr += "\r\n\r\n";

   endLatency = GetTickCount();

   c->write( (const char *) hdr, hdr.GetLength() );
   if( v == GET )
      c->write(buf, n);
}

// diagnostic
void GetRequest::eyeState()
{
   static char *typeStrs[] = {
      "Normal",
      "Test",
      "Barter",
      "Ian Dev" };


   CString hdr =
      "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: ";
   char buf[32000];
   *buf = 0;
   strstream text(buf, 32000, ios::out);

   // fill content
   text << "<html><body bgcolor=#ffffff>\r\n";
```



```
   text << "<h1>System State</h1><p>";
   text << "<table border=1 cellpadding=1>";
   text << "<tr><td><b>Name</b></td><td><b>Type</b></td><td><b>SI</b></td>";
   text << "<td><b>Ads Sent</b></td><td><b>Ads Booked</b></td></tr>\n";

   // Get a db connection to lock the ads array so that
   // it isn't reloaded or anything while we are processing.
   Database *db = getFromPool();

   for( int i = 0; i < ads.GetSize(); i++ ) {
      Ad *ad = ads.GetAt(i);
      text << "<tr><td><a href=\"http://ad.iantargets.com/viewad/";
      ad->fileName.MakeLower();
      text << ad->fileName << "\">" << ad->fileName << "</td>";
      text << "<td>" << typeStrs[ad->type] << "</td>";
      text << "<td>" << ad->si << "</td>";
      text << "<td>" << ad->nShown << "</td>";
      text << "<td>" << ad->maxImpressions << "</td></tr>\n";
   }

   releaseToPool(db);

   text << "</table>";
   text << "</body></html>";

   int n = text.pcount();
   char temp[100];
   itoa(n, temp, 10);   // content length
   hdr += temp;
   hdr += "\r\n\r\n";

   c->write( (const char *) hdr, hdr.GetLength() );
   c->write(buf, n);
}

// diagnostic
void GetRequest::whoAmI()
{
   Database &db = *getFromPool();

   User *user = User::lookupUser(db, userIP, request);
   user->lookupAncillaryInfo(db);

   CString hdr =
      "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: ";
   char buf[32000];
   *buf = 0;
   strstream text(buf, 32000, ios::out);

   // fill content
   text << "<html><body bgcolor=#ffffff><h1><IMG SRC=\"ianlogos.gif\" ALIGN=\"BOTTOM\">User In
   text << "<pre>";
   user->describe(db, text);
   text << "</pre></body></html>";

   int n = text.pcount();
   char temp[100];
   itoa(n, temp, 10);   // content length
   hdr += temp;
   hdr += "\r\n\r\n";

   c->write( (const char *) hdr, hdr.GetLength() );
   c->write(buf, n);

   delete user;
   releaseToPool(&db);
}

// diagnostic
void GetRequest::jumpingWhere(const char *from)
{
   ASSERT(FALSE);
   // fix for multi-db conns
/*
   User *user = User::lookupUser(userIP, request, FALSE);
```

```
        else {
            check(userAgent, "OmniWeb", brOmniWeb, osNEXT);
            check(userAgent, "Lynx", brLynx, osUnknown);
            check(userAgent, "IBM WebExplorer", brWebExplorer, osOS2);
            check(userAgent, "AIR_Mosaic", brAirMosaic, osWin);
            check(userAgent, "SPRY_Mosaic", brAirMosaic, osWin);
            check(userAgent, "MacWeb", brMacWeb, osMac);
            check(userAgent, "NetManage", brChameleon, osWin);
            check(userAgent, "Netsurfer", brNetsurfer, osNEXT);
            check(userAgent, "CNNworks", brCNN, osWin);
            check(userAgent, "interNotes", brNotes, osUnknown);
            check(userAgent, "Emissary", brEmissary, osUnknown);
            check(userAgent, "PipeMacWeb", brPipeMacWeb, osMac);
            check(userAgent, "InternetMCI", brMCI, osUnknown);
            check(userAgent, "Quarterdeck", brQuarterdeck, osUnknown);
            check(userAgent, "NCSA Mosaic (for the X ", brNCSA, osUnixUnknown);
            check(userAgent, "eworldbrowser", brEWorld, osMac) ) {
            if( check(userAgent, "68K") >= 0 )
                if( userAgent.Find("68K") >= 0 )
                    os = osMac68;
                else if( userAgent.Find("PPC") >= 0 )
                    os = osMacPPC;
                uniqueness = uNo;
                domainType = dtEWorld;
            }
            else if( check(userAgent, "PRODIGY", brProdigy, osUnknown) ) {
                uniqueness = uNo;
                domainType = dtProdigy;
            }
            else if( check(userAgent, "Delphi", brDelphi, osUnknown) ) {
                uniqueness = uNo;
                domainType = dtDelphi;
            }
            else if( browser == brUnknown ) {
                TRACE("unknown useragent: %s\n", (const char *) userAgent);
                liftOS(userAgent);
            }
        }

        if( userAgent.Find("via proxy") >= 0 ) {
            proxy = TRUE;
            if( uniqueness == uUnknown )
                uniqueness = uNo;
        }
    }
}
```

```
// location.cpp

#include "stdafx.h"
#include "objects.h"
#include "/d/toolkit/mapstate.h"
#include "/d/toolkit/tzutil.h"

// next line should be in tzutil.h
extern CountryTimeZoneMap mapCountryTimeZones;

struct IsDaylightSavings
{
   IsDaylightSavings()
   {
      TIME_ZONE_INFORMATION i;
      DWORD r = GetTimeZoneInformation(&i);
      daylightSavings = r == TIME_ZONE_ID_DAYLIGHT;
   }

   BOOL daylightSavings;
} ids;

tm* Location::userRelativeTime( time_t timeRelative )
{
   int utc_offset;
   int daylight_bias;

   if( country == 256 ) {
      if( !getStateTimeZoneInfo(state, utc_offset, daylight_bias) )
         return FALSE;
   }
   else if( country == 0 ) {
      return FALSE;
   }
   else {
      DWORD dwBias;
      if( !mapCountryTimeZones.Lookup(country, dwBias) )
         return FALSE;

      utc_offset = LOWORD(dwBias);
      daylight_bias = HIWORD(dwBias);
   }

   time_t ttTime;

   //
   // if timeRelative == 0, this assumes that they want the time
   // relative to the current time
   ttTime = timeRelative;
   if (!ttTime)
   {
      time(&ttTime);
   }

   if( ids.daylightSavings && daylight_bias != TZ_BIAS_UNDEFINED )
      ttTime += daylight_bias * 60 * 60;
   else
      ttTime += utc_offset * 60 * 60;

   return gmtime(&ttTime);
}
```

```
// request.h
//

#if !defined(_REQUEST_H_)
#define _REQUEST_H_

#include "d:/toolkit/sock.h"

enum Verb { UNKNOWN, GET, HEAD, POST };

class Connection;

class Request
{
public:
    Request(Connection *c, Verb v,
            const char *requestText,
            const sockaddr_in& from);

    virtual void service();

    DWORD getIP() const { return userIP; }
    const char* getRequest() const { return request; }
    Connection* getConn() const { return c; }

    void sendInternalError();

protected:
    BOOL sendFile(const char *fileName, const char *insertStr = 0);

    Connection *c;
    const char *request;
    Verb v;
    CString fileName;
    DWORD userIP;
};

void sendError(Connection *c, const char *msg, const char *headerField = 0);

#endif
```

```
// header.cpp
//

#include "stdafx.h"
#include "objects.h"
#include "/d/toolkit/laf_util.h"

const char cBrowser[] = "User-Agent:";

void message(const char *);

BOOL User::check(CString& userAgent, const char *pat, Browser b, OS o)
{
   if( browser != brUnknown )
      return FALSE;

   int l = strlen(pat);
   if( userAgent.Left(l) == pat ) {
      browser = b;
      os = o;
      const char *p = userAgent;
      p += l;
      p = strchr(p, '/');
      if( p ) {
         liftVer(p + 1);
      }
      return TRUE;
   }

   return FALSE;
}

static void match(OS& os, const char *userAgent, const char *pat, OS o)
{
   if( strstr(userAgent, pat) != 0 )
      os = o;
}

void User::liftOS(const CString& userAgent)
{
   if( userAgent.Find("X11") >= 0 ) {
      os = osUnixOther;
      match(os, userAgent, "SunOS", osUnixSun);
      match(os, userAgent, "HP-UX", osUnixHP);
      match(os, userAgent, "Linux", osUnixLinux);
      match(os, userAgent, "OSF", osUnixOSF1);
      match(os, userAgent, "AIX", osUnixAIX);
      match(os, userAgent, "IRIX", osUnixIRIX);
   }
   else if( userAgent.Find("Windows") >= 0 ) {
      if( userAgent.Find("32bit") >= 0 ||
          userAgent.Find("95") >= 0
        )
      {
         os = osWin32;
      } else {
         os = osWin16;
      }
   }
   else if( userAgent.Find("Win95") >= 0 ) {
      os = osWin95;
   }
   else if( userAgent.Find("Win16") >= 0 ) {
      os = osWin16;
   }
   else if( userAgent.Find("Macintosh") >= 0 ) {
      os = osMac;
      match(os, userAgent, "PPC", osMacPPC);
      match(os, userAgent, "68K", osMac68);
   }
   else if( userAgent.Find("WinNT") >= 0 ) {
      os = osWinNT;
   }
   else {
```



```
      s << userAgent.Left(70);
      message(s);
   }
}

// derive information about the user from the request header
//
void User::headerDerive(const char *requestHeader)
{
   const char *ua = strstr(requestHeader, cBrowser);
   if( ua == 0 ) {
      // if no user agent field, something weird we
      // don't know much about, don't assume unique.
      uniqueness = uUnlikely;
   }
   else {
      ua += 11;
      while( *ua == ' ' )
         ua++;
      const char *p = strchr(ua, '\r');
      if( p ) {
         CString userAgent(ua, p - ua);
         if( userAgent.Left(8) == "Mozilla/" ) {
            browser = brNetscape;
            liftVer((const char *) userAgent + 8);

            // OS
            liftOS(userAgent);
         }
         else if( userAgent.Left(12) == "NCSA Mosaic/" ) {
            browser = brNCSA;
            liftVer((const char *) userAgent + 12);

            // OS
            match(os, userAgent, "Windows", osWin);
            match(os, userAgent, "X11", osUnixUnknown);
            match(os, userAgent, "X Window", osUnixUnknown);
            match(os, userAgent, "IWENG/", 6) == 0 ) {
         }
         else if( strncmp(userAgent,
            browser = brAOL;
            uniqueness = uNo;
            domainType = dtAOL;
            liftVer((const char *) userAgent + 6);
            os = osWin;
         }
         else if( strncmp(userAgent, "aolbrowser/", 10) == 0 ) {
            browser = brAOL;
            uniqueness = uNo;
            domainType = dtAOL;
            liftVer((const char *) userAgent + 11);
            os = osMac;
         }
         else if( userAgent.Left(28) == "Microsoft Internet Explorer/" ) {
            // Microsoft Internet Explorer/4.40
            browser = brMicrosoft;
            liftVer((const char *) userAgent + 28);
            os = osWin32;
            match(os, userAgent, "Windows 95", osWin95);
         }
         else if( userAgent.Left(8) == "HotJava/" ) {
            browser = brHotJava;
            liftVer((const char *) userAgent + 8);
         }
         else if( userAgent.Left(16) == "Enhanced_Mosaic/" ) {
            browser = brEnhancedMosaic;
            liftVer((const char *) userAgent + 16);
            os = osWin;
            if( userAgent.Find("Win32") >= 0 )
               os = osWin32;
         }
         else if( userAgent.Left(11) == "NetCruiser/" ) {
            liftVer((const char *) userAgent + 11);
            browser = brNetCruiser;
            os = osWin;
         }
```

SERVER.H 23-Sep-1995 15:30 Page 1(1)

```
// server.h
// General ad server startup stuff.
//

BOOL startServer();
```

HIGHLY CONFIDENTIAL

DC 069486

```
// status.h

void setStatus(const char *s);

extern int adsSent;
extern int jumpsTaken;
extern int totalAdSendLatency;
extern int totalAdSendTime;
extern int timeOuts;
extern int poolTimeOuts;

extern int barter, lanDev, testAd;

void latencyWas(int n);
void adSendTimeWas(int n);

void adSent();
```

EXHIBIT B

```
REQUEST.H                    11-Jan-1996 11:35                    Page 1(1)

getrequest.h

f !defined(_GETREQUEST_H_)
efine _GETREQUEST_H_

nclude "request.h"
nclude "objects.h"

ass GetRequest : public Request

blic:
  GetRequest(Connection *c, Verb v,
             const char *requestText,
             const sockaddr_in& from) :
                 Request(c, v, requestText, from) { }

  virtual void service();

otected:
  void whoAmI();
  void jumpingWhere(const char *from);
  void sendAd(const char *from);
  void activity(const char *activityStr);     // Netscape 2.0 frames
  void sendFrame(const char *from);
  void takeJump(const char *from);
  void sysStats();

  void sendIDatabase& db, Ad *ad, User *u);

  // send info
  void sendInfo(const char *url);
  void s1_(const char *url);

endif
```

DX 50

HIGHLY CONFIDENTIAL

DC 069484

```
// rememberad.h
//

void rememberSend(Ad *ad, User *u, const char *fromDoc);

// returns Ad ID
DWORD queryAdSent(User *u, const char *fromDoc);
```